Regession and Classification with Framingham Heart Data

Sally Austen

2023-11-30

Set Up Environment

```
# Clear global environment
rm(list = ls())

# Set up libraries
# suppressMessage and suppressWarning functions used to make R-Markdown look better (not
↪  necessary)
suppressMessages(suppressWarnings(library(dplyr)))
suppressMessages(suppressWarnings(library(ggcorrplot)))
suppressMessages(suppressWarnings(library(factoextra)))
suppressMessages(suppressWarnings(library(glmnet)))
suppressMessages(suppressWarnings(library(tidyverse)))
suppressMessages(suppressWarnings(library(pls)))
suppressMessages(suppressWarnings(library(DAAG)))
suppressMessages(suppressWarnings(library(FNN)))
suppressMessages(suppressWarnings(library(class)))
suppressMessages(suppressWarnings(library(caret)))

# To create reproducible results
set.seed(123)

# Read in csv to framingham data set
framingham <- read.csv("data/frm.csv")
head(framingham) # Preview data frame
```

```
##   X RANDID SEX TOTCHOL AGE SYSBP DIABP CURSMOKE CIGPDAY   BMI DIABETES BPMEDS
## 1 1   2448   1     195  39 106.0  70.0        0       0 26.97        0      0
## 2 2   2448   1     209  52 121.0  66.0        0       0    NA        0      0
## 3 3   6238   2     250  46 121.0  81.0        0       0 28.73        0      0
## 4 4   6238   2     260  52 105.0  69.5        0       0 29.43        0      0
## 5 5   6238   2     237  58 108.0  66.0        0       0 28.50        0      0
## 6 6   9428   1     245  48 127.5  80.0        1      20 25.34        0      0
##   HEARTRTE GLUCOSE educ PREVCHD PREVAP PREVMI PREVSTRK PREVHYP TIME PERIOD HDLC
## 1       80      77    4       0      0      0        0       0    0      1   NA
## 2       69      92    4       0      0      0        0       0 4628      3   31
## 3       95      76    2       0      0      0        0       0    0      1   NA
## 4       80      86    2       0      0      0        0       0 2156      2   NA
## 5       80      71    2       0      0      0        0       0 4344      3   54
## 6       75      70    1       0      0      0        0       0    0      1   NA
```

```
##   LDLC DEATH ANGINA HOSPMI MI_FCHD ANYCHD STROKE CVD HYPERTEN TIMEAP TIMEMI
## 1   NA     0      0      1       1      1      0   1        0   8766   6438
## 2  178     0      0      1       1      1      0   1        0   8766   6438
## 3   NA     0      0      0       0      0      0   0        0   8766   8766
## 4   NA     0      0      0       0      0      0   0        0   8766   8766
## 5  141     0      0      0       0      0      0   0        0   8766   8766
## 6   NA     0      0      0       0      0      0   0        0   8766   8766
##   TIMEMIFC TIMECHD TIMESTRK TIMECVD TIMEDTH TIMEHYP
## 1     6438    6438     8766    6438    8766    8766
## 2     6438    6438     8766    6438    8766    8766
## 3     8766    8766     8766    8766    8766    8766
## 4     8766    8766     8766    8766    8766    8766
## 5     8766    8766     8766    8766    8766    8766
## 6     8766    8766     8766    8766    8766    8766
```

Regression Problem: Predict BMI

Set up for regression: modify the data set and set X and y

```
# Create new data frame with only risk factor data (index 1-24)
# Don't include HDLC and LDLC (index 23, 24) data due to high number of NA's
fram_regression <- framingham %>% select(1:22)
# Look at modified data frame
head(fram_regression)
```

```
##   X RANDID SEX TOTCHOL AGE SYSBP DIABP CURSMOKE CIGPDAY   BMI DIABETES BPMEDS
## 1 1   2448   1     195  39 106.0  70.0        0       0 26.97        0      0
## 2 2   2448   1     209  52 121.0  66.0        0       0    NA        0      0
## 3 3   6238   2     250  46 121.0  81.0        0       0 28.73        0      0
## 4 4   6238   2     260  52 105.0  69.5        0       0 29.43        0      0
## 5 5   6238   2     237  58 108.0  66.0        0       0 28.50        0      0
## 6 6   9428   1     245  48 127.5  80.0        1      20 25.34        0      0
##   HEARTRTE GLUCOSE educ PREVCHD PREVAP PREVMI PREVSTRK PREVHYP TIME PERIOD
## 1       80      77    4       0      0      0        0       0    0      1
## 2       69      92    4       0      0      0        0       0 4628      3
## 3       95      76    2       0      0      0        0       0    0      1
## 4       80      86    2       0      0      0        0       0 2156      2
## 5       80      71    2       0      0      0        0       0 4344      3
## 6       75      70    1       0      0      0        0       0    0      1
```

```
# Make new variables
#   - past_bmi = last measured bmi
#   - last_exam = time since last exam
fram_regression <- fram_regression %>%
  group_by(RANDID) %>% # Group by patient
  mutate(past_bmi = lag(BMI, default = NA), # Get BMI from last exam
         last_exam = TIME - lag(TIME, default = first(TIME))) %>% # Calculate number of
         ↪  days since last exam
  ungroup()

# Look at amount of NA values for each variable
```

```
nacols<-apply(fram_regression, 2, function(x) {sum(is.na(x))})
nacols
```

```
##         X    RANDID       SEX   TOTCHOL       AGE     SYSBP     DIABP  CURSMOKE
##         0         0         0       409         0         0         0         0
##   CIGPDAY       BMI  DIABETES    BPMEDS  HEARTRTE   GLUCOSE      educ   PREVCHD
##        79        52         0       593         6      1440       295         0
##    PREVAP    PREVMI  PREVSTRK   PREVHYP      TIME    PERIOD  past_bmi last_exam
##         0         0         0         0         0         0      4453         0
```

```
# Remove columns that are unnecessary and/or have lots of missing data
fram_regression <- fram_regression[, !(colnames(fram_regression) %in% c("X", "RANDID",
↪  "PERIOD", "GLUCOSE", "TIME", "BPMEDS", "CIGPDAY"))]

# Drop all rows with NA entries (will drop all period 1 data points and more)
fram_regression <- na.omit(fram_regression)

# Put data set into matrix form
data <- as.matrix(fram_regression)

# Set X -> a matrix made up of the risk factors being studied
X <- data[, -7]
head(X) # Preview matrix
```

```
##      SEX TOTCHOL AGE SYSBP DIABP CURSMOKE DIABETES HEARTRTE educ PREVCHD PREVAP
## [1,]   2     260  52   105  69.5        0        0       80    2       0      0
## [2,]   2     237  58   108  66.0        0        0       80    2       0      0
## [3,]   1     283  54   141  89.0        1        0       75    1       0      0
## [4,]   2     232  67   183 109.0        1        0       60    3       0      0
## [5,]   2     343  51   109  77.0        1        0       90    3       0      0
## [6,]   2     230  49   177 102.0        0        0      120    2       0      0
##      PREVMI PREVSTRK PREVHYP past_bmi last_exam
## [1,]      0        0       0    28.73      2156
## [2,]      0        0       0    29.43      2188
## [3,]      0        0       0    25.34      2199
## [4,]      0        0       1    28.58      1977
## [5,]      0        0       0    23.10      2072
## [6,]      0        0       1    30.30      2178
```

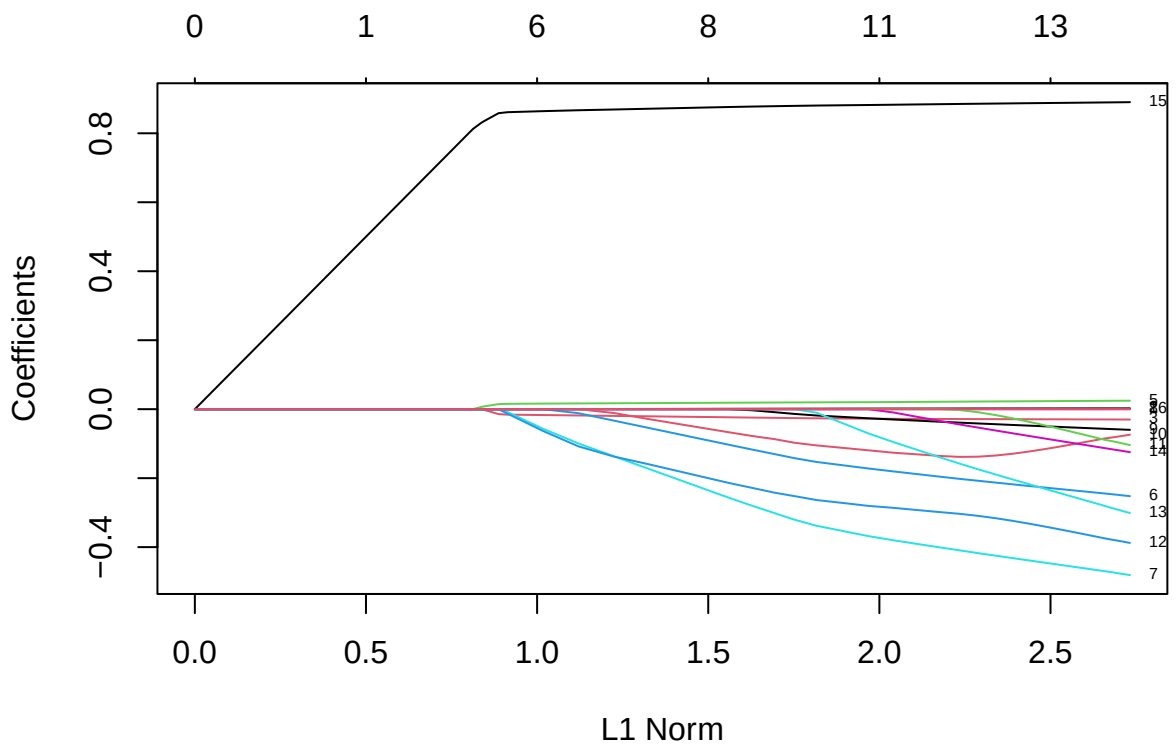
```
# Set Y -> column vector of BMI values
y <- data[, 7]
head(y) # Preview matrix
```

```
## [1] 29.43 28.50 25.34 30.18 23.48 31.36
```

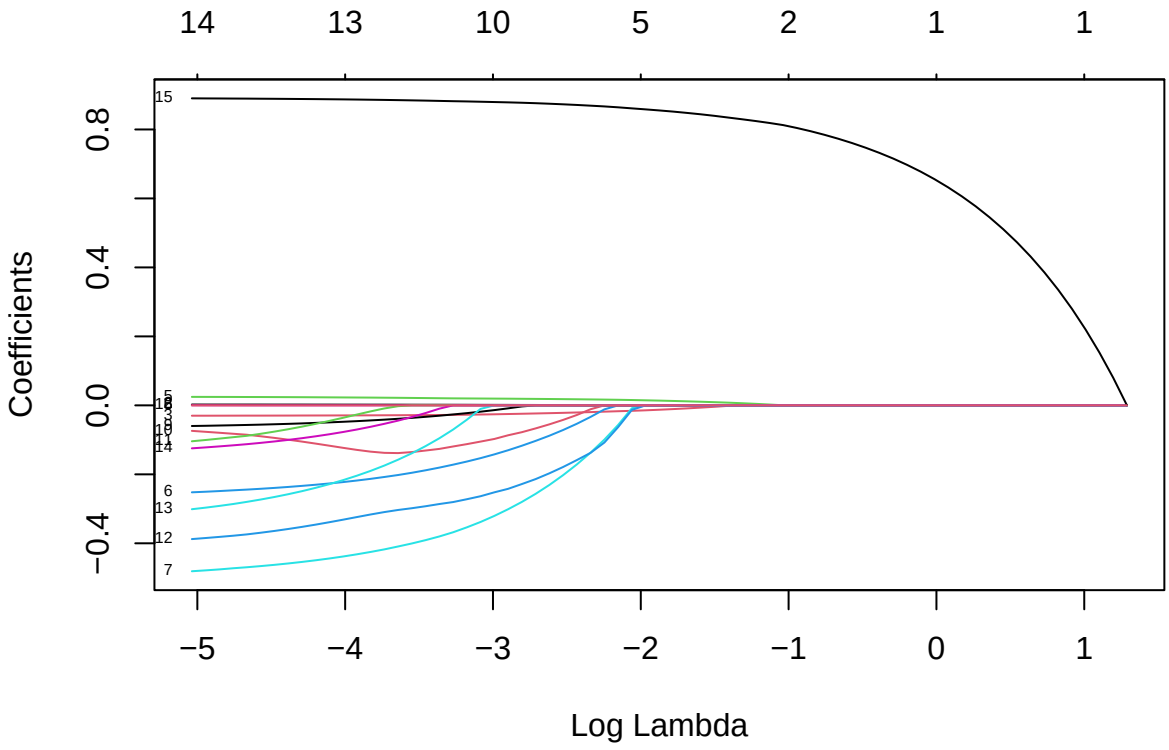
Lasso and Variable Selection

```
# Set alpha to 1 for Lasso
alpha <- 1
```

```r
# Use lasso to fit the data
fitL = glmnet(X,y, alpha = alpha)
# Plot variable coefficients vs L1 norm (the penalty applied for complexity)
plot(fitL, label=T)
```



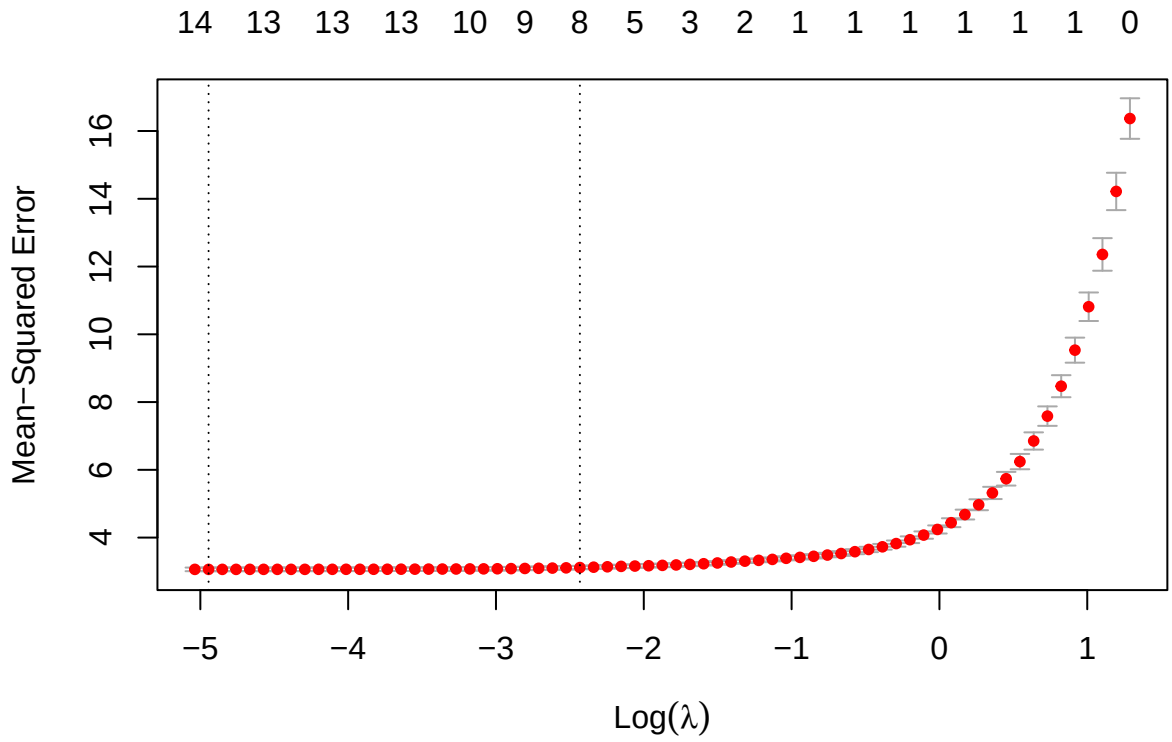

```r
# Plot variable coefficients vs log(lambda)
plot(fitL, xvar = "lambda", label=T)
```




```
# Print the variable names and corresponding numbers to better analyze graph
var_names <- colnames(X)
var_numbers <- 1:length(var_names)
print(data.frame(Variable_Number = var_numbers, Variable_Name = var_names))
```

```
##    Variable_Number Variable_Name
## 1                1           SEX
## 2                2       TOTCHOL
## 3                3           AGE
## 4                4         SYSBP
## 5                5         DIABP
## 6                6      CURSMOKE
## 7                7      DIABETES
## 8                8      HEARTRTE
## 9                9          educ
## 10              10       PREVCHD
## 11              11        PREVAP
## 12              12        PREVMI
## 13              13      PREVSTRK
## 14              14       PREVHYP
## 15              15      past_bmi
## 16              16     last_exam
```

```r
# 10-fold cross-validation for lambda values
cvfitL = cv.glmnet(X,y, alpha = alpha)
# Plot MSE vs log(lambda) to find best lambda
plot(cvfitL)
```




```r
# Lambda that minimizes the cross-validated error
cat("Lamda that minimizes error: ", cvfitL$lambda.min, "\n")
```

```
## Lamda that minimizes error:  0.007124052
```

```r
# Lambda that is one standard deviation away from minimum
cat("Lamda that gives best model: ", cvfitL$lambda.1se, "\n")
```

```
## Lamda that gives best model:  0.08782864
```

```r
# Set lambda to value 1 standard deviation away from minimum
best_lambda <- cvfitL$lambda.1se
# Create the model with best lambda value
modelL <- glmnet(X, y, alpha = alpha, lambda = best_lambda)
# Print the coefficients for the LASSO model
coef(modelL)
```

```
## 17 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  2.988335783
## SEX          .
## TOTCHOL      0.000627423
## AGE         -0.020643698
## SYSBP        .
## DIABP        0.017629449
## CURSMOKE    -0.053331511
## DIABETES    -0.171090761
## HEARTRTE     .
## educ         .
## PREVCHD     -0.030687583
## PREVAP       .
## PREVMI      -0.156057110
## PREVSTRK     .
## PREVHYP      .
## past_bmi     0.871050538
## last_exam    .
```

Evaluate LASSO model by doing 10-fold cross validation and finding the average error

```r
# Set the number of folds for cross-validation (e.g., 10 folds)
num_folds <- 10

# Create an index vector for cross-validation
folds <- sample(1:num_folds, size = nrow(X), replace = TRUE)

# Initialize vectors to store training and testing errors for each fold
train_errors <- numeric(num_folds)
test_errors <- numeric(num_folds)

# Perform k-fold cross-validation
for (fold in 1:num_folds) {
  # Split the data into training and testing sets
  train_data <- subset(X, folds != fold)
  test_data <- subset(X, folds == fold)
  train_labels <- y[folds != fold]
  test_labels <- y[folds == fold]

  # Train the LASSO model on the training data
  modelL_fold <- glmnet(train_data, train_labels, alpha = alpha, lambda = best_lambda)

  # Make predictions on the training set
  train_pred <- predict(modelL_fold, newx = train_data)

  # Make predictions on the testing set
  test_pred <- predict(modelL_fold, newx = test_data)

  # Calculate training error (e.g., Mean Squared Error)
  train_errors[fold] <- mean((train_pred - train_labels)^2)
```

```
  # Calculate testing error (e.g., Mean Squared Error)
  test_errors[fold] <- mean((test_pred - test_labels)^2)
}

# Calculate average training and testing errors
avg_train_error <- mean(train_errors)
avg_test_error <- mean(test_errors)

# Print the results (Residual Standard Errors)
cat("Average LASSO Training Error:", avg_train_error, "\n")
```

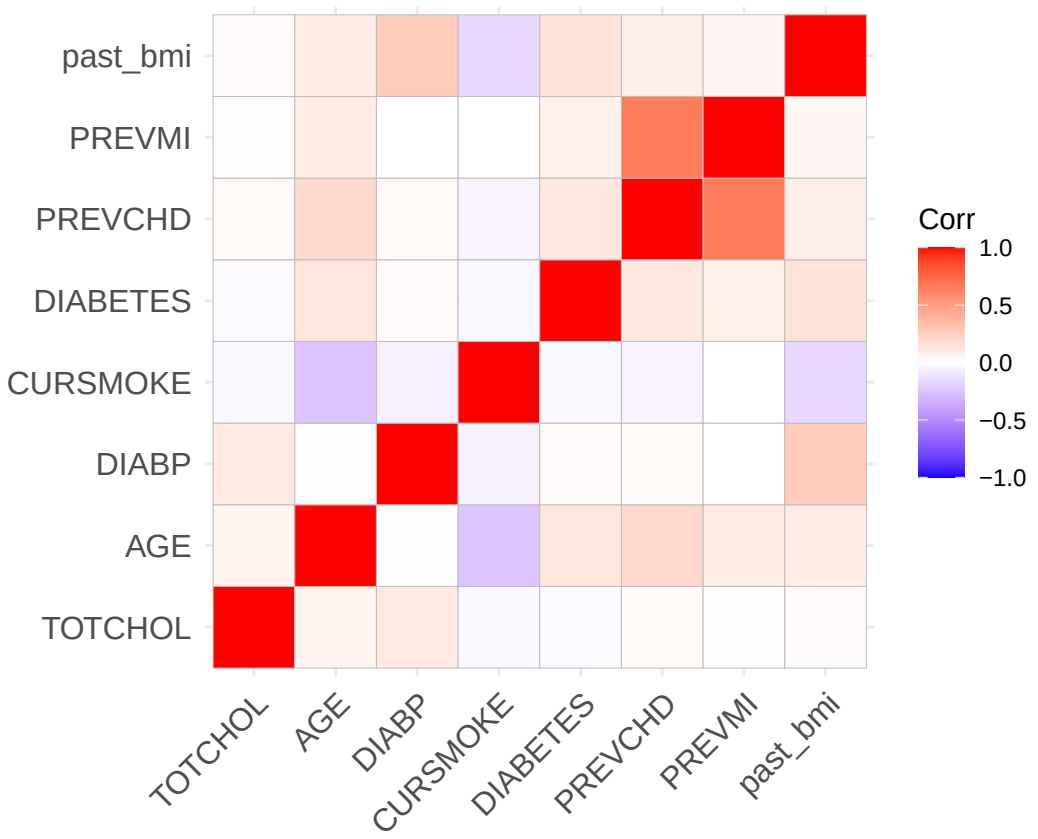
```
## Average LASSO Training Error: 3.100927
```

```
cat("Average LASSO Testing Error:", avg_test_error, "\n")
```

```
## Average LASSO Testing Error: 3.114735
```

PCA and dimension reduction

```
# Select variables based on LASSO (exclude variables whose coefficients went to 0)
new_fram <- fram_regression[, c("TOTCHOL", "AGE", "DIABP", "CURSMOKE", "DIABETES",
↪  "PREVCHD", "PREVMI", "past_bmi", "BMI")]

# X includes: TOTCHOL, AGE, DIABP, CURSMOKE, DIABETES, PREVCHD, PREVMI, past_bmi
new_X <- new_fram[, !(colnames(new_fram) %in% c("BMI"))]

# Normalize data for PCA
X_normalized <- scale(new_X)

# Compute correlation matrix
corr_matrix <- cor(X_normalized)
# Graph correlation matrix
ggcorrplot(corr_matrix)
```




```
# Apply PCA
data.pca <- princomp(corr_matrix)
# Look at potential dimension reduction by analyzing the cumulative proportion
summary(data.pca)
```

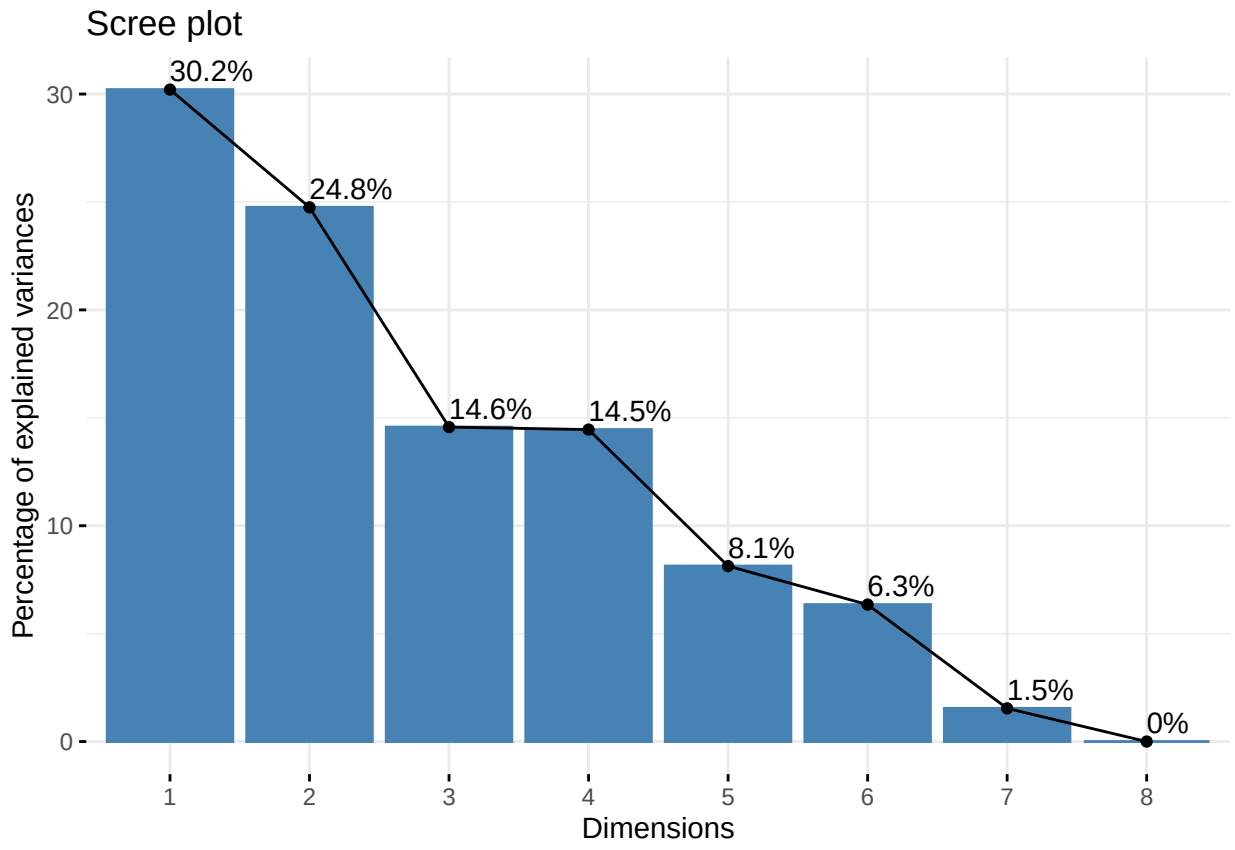
```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3    Comp.4     Comp.5
## Standard deviation     0.5258093 0.4759743 0.3651740 0.3637350 0.27279653
## Proportion of Variance 0.3020726 0.2475266 0.1456983 0.1445522 0.08130784
## Cumulative Proportion  0.3020726 0.5495992 0.6952974 0.8398497 0.92115749
##                           Comp.6     Comp.7       Comp.8
## Standard deviation     0.24100796 0.11864520 7.933578e-09
## Proportion of Variance 0.06346255 0.01537996 6.876903e-17
## Cumulative Proportion  0.98462004 1.00000000 1.000000e+00
```

```
# Take a look into how the variables are represented in the components
# Components 1-5 explain 92% of variance in data
data.pca$loadings[, 1:5]
```

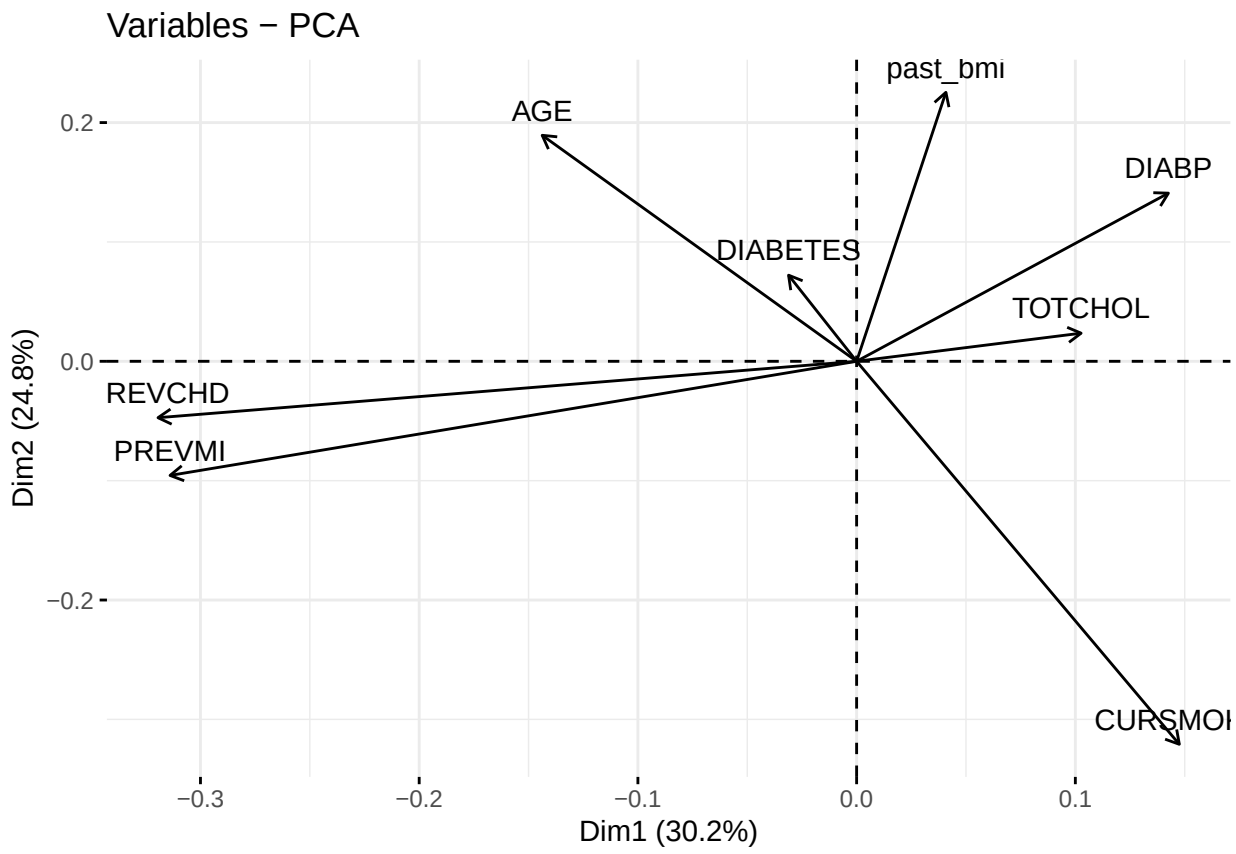
```
##               Comp.1      Comp.2      Comp.3       Comp.4        Comp.5
## TOTCHOL   0.19518967  0.04946252  0.72475925  0.172647251  0.5068383100
## AGE      -0.27340019  0.39776597  0.13991145  0.494672832 -0.5594327142
## DIABP     0.27108948  0.29593293  0.05222311 -0.569759690 -0.0494549088
## CURSMOKE  0.28058181 -0.67413356 -0.18129266  0.002906849 -0.2377085066
```

```
## DIABETES -0.05918251  0.15156977 -0.58820536  0.376022855  0.5865085666
## PREVCHD  -0.60744575 -0.09928050  0.07182084 -0.229437960  0.0891827275
## PREVMI   -0.59700837 -0.20089175  0.06364138 -0.278057990  0.1387534149
## past_bmi  0.07762581  0.47340874 -0.25369376 -0.359869353 -0.0006898801
```

```r
# Plot the eigenvalues/components against the number of dimensions
fviz_eig(data.pca, addlabels = TRUE)
```




```r
# Graph of the variables and how they are related to the dimensions
#   - Direction implies variable correlation
#   - Magnitude implies variable's impact on component
fviz_pca_var(data.pca, col.var = "black")
```




```
# TODO: Cross-validation to pick num of components
# Set number of components
num_comps <- 5

# Extract the first 5 principal components
princ_comps <- predict(data.pca, newdata = X_normalized)[, 1:num_comps]
# Combine the principal components with other predictors
bmi_and_pca <- cbind(new_fram[, 9], princ_comps)

# Perform linear regression using the first 5 components
modelPCA <- lm(BMI ~ ., data = bmi_and_pca)
summary(modelPCA)
```

```
##
## Call:
## lm(formula = BMI ~ ., data = bmi_and_pca)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.3762  -1.8128  -0.0728   1.7173  13.8921
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 26.20883    0.03539 740.583   <2e-16 ***
## Comp.1       0.36224    0.02699  13.420   <2e-16 ***
```

```
## Comp.2       1.83853    0.02911  63.160   <2e-16 ***
## Comp.3      -0.76408    0.03363 -22.722   <2e-16 ***
## Comp.4      -1.67892    0.03323 -50.528   <2e-16 ***
## Comp.5       0.32263    0.03792   8.508   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.784 on 6629 degrees of freedom
## Multiple R-squared:  0.5287, Adjusted R-squared:  0.5283
## F-statistic:  1487 on 5 and 6629 DF,  p-value: < 2.2e-16
```

Compare Models

```
# Compare regression models
cat("RSE for LASSO:", avg_train_error, "\n")
```

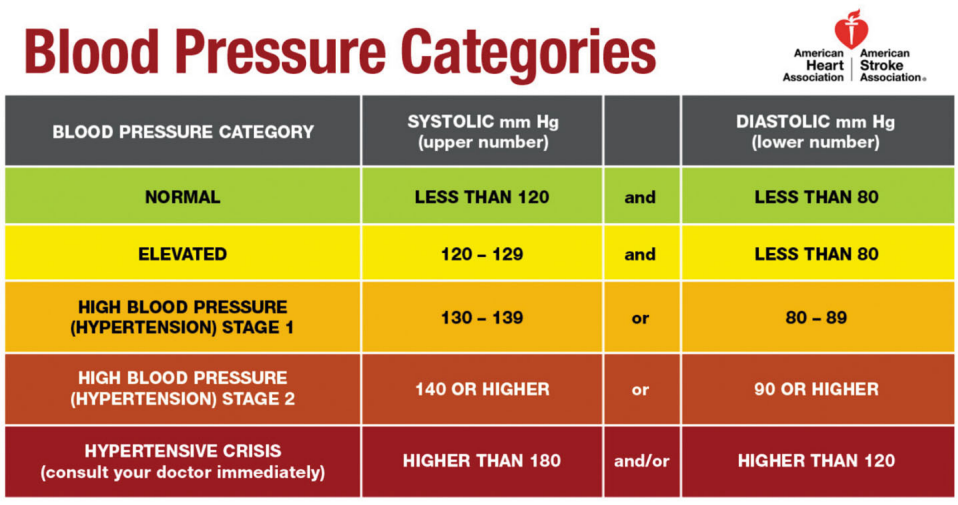
```
## RSE for LASSO: 3.100927
```

```
cat("RSE for PCA Method:", summary(modelPCA)$sigma, "\n")
```

```
## RSE for PCA Method: 2.783848
```

Classification Problem: Predict blood pressure category

```
# Put blood pressure categories into R-markdown file
knitr::include_graphics("Rendering/bp_cats.jpg")
```



Set up for classification: modify the data set, set X and y, and create loss function

```r
# Function to turn blood pressure numbers into categories
# Normal = 0; Elevated = 1; Stage 1 = 2; Stage 2 = 3; Crisis = 4
categorize <- function(SYSBP, DIABP) {
  if (is.na(SYSBP) | is.na(DIABP)){
    return(NA)
  } else if (SYSBP > 180 | DIABP > 120) {
    # Crisis
    return(5)
  } else if (SYSBP >= 140 | DIABP >= 90) {
    # Stage 2
    return(4)
  } else if (SYSBP >= 130 | DIABP >= 80) {
    # Stage 1
    return(3)
  } else if (SYSBP >= 120){
    # Elevated
    return(2)
  } else {
    # Normal
    return(1)
  }
}

# Make new variable for blood pressure category
fram_classification <- framingham %>%
  rowwise() %>% # Iterate through each row
  mutate(BP_CAT = categorize(SYSBP, DIABP)) # Put each person into blood type categories

# Focus on SEX, BMI, AGE, DIABETES, HEARTRATE
wanted_vars <- c("SEX", "BMI", "AGE", "DIABETES", "HEARTRTE", "BP_CAT")
# Narrow down data set to desired variables
fram_classification <- fram_classification %>%
  select(all_of(wanted_vars))

# Look at number of NA's
nacols <- apply(fram_classification, 2, function(x) {sum(is.na(x))})
nacols
```

```
##      SEX      BMI      AGE DIABETES HEARTRTE   BP_CAT
##        0       52        0        0        6        0
```

```r
# Drop all rows with NA entries
fram_classification <- na.omit(fram_classification)

# Put data frame into matrix form
data <- as.matrix(fram_classification)

# X -> matrix with x variables
X <- data[, -6]
head(X)
```

```
##      SEX   BMI AGE DIABETES HEARTRTE
```

```
## [1,]   1 26.97  39        0       80
## [2,]   2 28.73  46        0       95
## [3,]   2 29.43  52        0       80
## [4,]   2 28.50  58        0       80
## [5,]   1 25.34  48        0       75
## [6,]   1 25.34  54        0       75
```

```r
# y -> column vector with blood pressure categories
y <- data[, 6]
head(y)
```

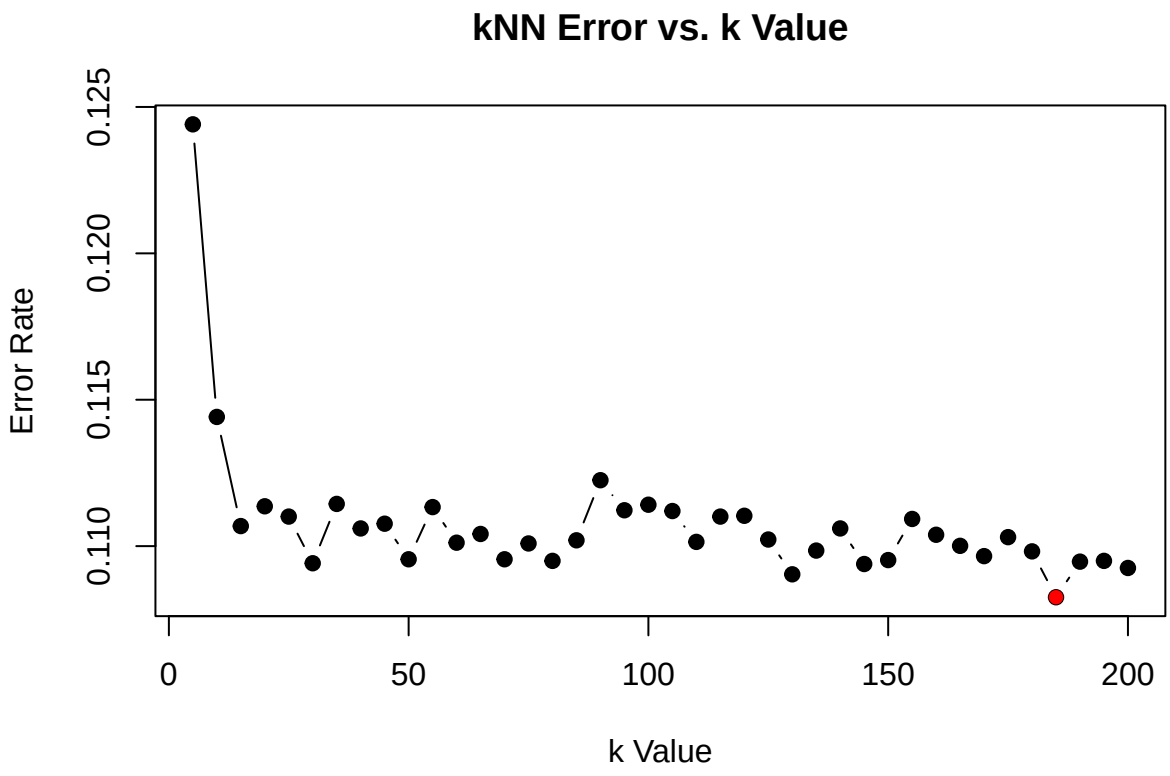
```
## [1] 1 3 1 1 3 4
```

```r
# Loss function that penalizes more for worse classifications
# Parameters: predicted and observed vectors of same length
# Returns mean of the squared difference (normalized)
loss <- function(predicted, observed) {
  # Get difference between predicted and observed classes
  diff <- predicted - observed

  # Calculate squared differences between predicted and observed classes
  # Gives higher punishments for classifications further away from observed
  squared_diff <- (diff)^2

  # Calculate mean squared difference
  mean_squared_diff <- mean(squared_diff)

  # Normalize the error by dividing by the greatest possible penalty
  normalized_error <- mean_squared_diff / 16

  # Return the normalized error
  return(normalized_error)
}
```

Use KNN for classificaiton

```r
# Select index to split data into train and test data sets
validationIndex <- createDataPartition(fram_classification$BP_CAT, p=0.80, list=FALSE)

train <- data[validationIndex,] # 80% of data to training
test <- data[-validationIndex,] # Remaining 20% for test

# Scale train and test subsets
train_scale <- scale(as.matrix(train))
test_scale <- scale(test)

# Create X and y for training subset
Xtr_scale <- as.matrix(train_scale[,-6])
Ytr <- as.matrix(train[, 6])
```

```r
# Create X and y for testing subset
Xte_scale <- as.matrix(test_scale[, -6])
Yte <- as.matrix(test[, 6])

# Create empty vectors to store k values and corresponding errors
k_values <- seq(5, 200, by = 5)
errors <- numeric(length(k_values))

# Loop through each k value
for (i in seq_along(k_values)) {
  k_value <- k_values[i]

  # Train the kNN model
  modelKNN <- knn(
    train = Xtr_scale,
    test = Xte_scale,
    cl = Ytr,
    k = k_value
  )

  # Get predicted vector
  predicted <- as.numeric(modelKNN)

  # Calculate error with custom loss function
  error <- loss(predicted, Yte)

  # Store the normalized error
  errors[i] <- error
}

# Find the minimum error value and its corresponding k value
error_KNN <- min(errors)
optimal_k <- k_values[which.min(errors)]

# Plot the relationship between k values and errors
plot(k_values, errors, type = "b", pch = 19,
     xlab = "k Value", ylab = "Error Rate", main = "kNN Error vs. k Value")

# Highlight the point with the minimum error
points(optimal_k, error_KNN, col = "red", pch = 16)
```




```
cat("Minimium error: ", error_KNN, "\n")
```

```
## Minimium error:  0.1082541
```

```
cat("k value with lowest error (optimal k): ", optimal_k, "\n")
```

```
## k value with lowest error (optimal k):  185
```

Use One-Hot Encoding for classification

```
# Used the function from in class material
# Sets up one hot encoding by manipulating y
OneHot <- function(g, cls) {
    Y <- as.numeric(g==cls[1])
    for (b in cls[-1]){
        y <- as.numeric(g==b)
        Y <- cbind(Y,y)
    }
    return(Y)
}

# Put categories into an array
```

```r
cls <- as.array(c(1, 2, 3, 4, 5))

# The correct classifications
g <- y

# Change Y into an array of 1's and 0's
Y <- OneHot(g, cls)
head(Y)
```

```
##      Y y y y y
## [1,] 1 0 0 0 0
## [2,] 0 0 1 0 0
## [3,] 1 0 0 0 0
## [4,] 1 0 0 0 0
## [5,] 0 0 1 0 0
## [6,] 0 0 0 1 0
```

```r
# Manipulate X to allow for matrix operations
X <- apply(X, 2, as.numeric)
head(X)
```

```
##      SEX   BMI AGE DIABETES HEARTRTE
## [1,]   1 26.97  39        0       80
## [2,]   2 28.73  46        0       95
## [3,]   2 29.43  52        0       80
## [4,]   2 28.50  58        0       80
## [5,]   1 25.34  48        0       75
## [6,]   1 25.34  54        0       75
```

```r
# Compute beta
XTX <- t(X) %*% X
XTY <- t(X) %*% Y
B <- solve(XTX,XTY)
B
```

```
##                     Y             y            y             y             y
## SEX       0.112086519  0.0239344380 -0.021414733 -0.0680385707  0.0045315543
## BMI      -0.001063998 -0.0023139175  0.008178176  0.0074354195  0.0007866079
## AGE      -0.001863021  0.0017257325 -0.001928117  0.0057954621  0.0012757547
## DIABETES -0.088443189 -0.0344452199 -0.097177543  0.0682084611  0.0916427711
## HEARTRTE  0.001784032  0.0002684484  0.002757664 -0.0001906893 -0.0006781160
```

```r
# Compute Yhat
Yhat <- X%*%B

# Give the index of the greatest element in each vector
ghat <- apply(Yhat,1,which.max)

# Index corresponds with class, so predicted values are the index of the max value in
↪  each row of Y
predicted <- ghat
```

```r
# Get errors calculated by custom loss function
error_OH <- loss(predicted, g)

cat("Error: ", error_OH, "\n")
```

```
## Error:  0.1381502
```

Compare Models

```r
# Compare classification models
cat("Error for KNN:", error_KNN, "\n")
```

```
## Error for KNN: 0.1082541
```

```r
cat("Error for One-Hot Encoding:", error_OH, "\n")
```

```
## Error for One-Hot Encoding: 0.1381502
```